Concurrency Technicals & Dynamic Management View

Team ENJOY!!

Assignment 11

Chukwuebeka Ezema
Neti Sheth
Josh Stamper
Yagna Venkitasamy

TABLE OF CONTENTS

Isolation Levels:

Isolation determines how transaction integrity is visible to other users and systems. It means that a transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a database system.
Isolation levels define the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system. A transaction isolation level is defined by the following phenomena –

Dirty Read – A Dirty read is the situation when a transaction reads a data that has not yet been committed. For example, Let's say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.

Non-Repeatable Read – Non-Repeatable read occurs when a transaction reads same row twice and get a different value each time. For example, suppose transaction T1 reads data. Due to concurrency, another transaction T2 updates the same data and commit, now if transaction T1 rereads the same data, it will retrieve a different value.

Phantom Read – Phantom Read occurs when two same queries are executed, but the rows retrieved by the two, are different. For example, suppose transaction T1 retrieves a set of rows that satisfy some search criteria. Now, Transaction T2 generates some new rows that match the search criteria for transaction T1. If transaction T1 re-executes the statement that reads the rows, it gets a different set of rows this time.

Locking Options:

Each transaction requests locks of different types on the resources, such as rows, pages, or tables, on which the transaction is dependent. The locks block other transactions from modifying the resources in a way that would cause problems for the transaction requesting the lock. Each transaction frees its locks when it no longer has a dependency on the locked resources.

1. Exclusive lock (X) – This lock type, when imposed, will ensure that a page or row will be reserved *exclusively* for the transaction that imposed the exclusive lock as long as the transaction holds the lock.

The exclusive lock will be imposed by the transaction when it wants to modify the page or row data, which is in the case of DML statements DELETE, INSERT and UPDATE. An exclusive lock can be imposed to a page or row only if there is no other shared or exclusive lock imposed already on the target. This practically means that only one exclusive lock can be imposed to a page or row, and once imposed no other lock can be imposed on locked resources. Features include:

- Used for DML operations
- Prevents other users from accessing the resource.
- Operations, such as INSERT, UPDATE, or DELETE means DML query. Ensures that multiple updates cannot be made to the same resource at the same time.

2. <u>Shared lock (S)</u> – This lock type, when imposed, will reserve a page or row to be available only for reading, which means that any other transaction will be prevented to modify the locked record as long as the lock is active. However, a shared lock can be imposed by several transactions at the same time over the same page or row and in that way several transactions can *share* the ability for data reading since the reading process itself will not affect anyhow the actual page or row data. In addition, a shared lock will allow write operations, but no DDL changes will be allowed. Features include:

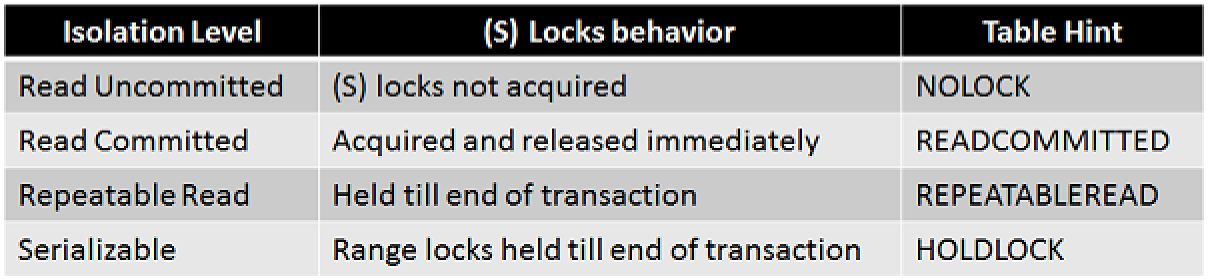
- Used for SELECT operations
- Enable other sessions to perform select operations but prevent updates
- read-only operations
- Operation with SELECT statement generally use in Shared mode.

3.<u>Update lock (U)</u> – This lock is like an exclusive lock but is designed to be more flexible in a way. An update lock can be imposed on a record that already has a shared lock. In such a case, the update lock will impose another shared lock on the target row. Once the transaction that holds the update lock is ready to change the data, the update lock (U) will be transformed to an exclusive lock (X). It is important to understand that update lock is asymmetrical in regards of shared locks. While the update lock can be imposed on a record that has the shared lock, the shared lock cannot be imposed on the record that already has the update lock.

- Preliminary stage for exclusive lock. Used by the server when filtering the records to be modified
- Prevents other update locks
- A solution to the cycle deadlock problem

In short, we summarize that:

- (S) locks are compatible with (S) and (U) locks.
- (X) locks are incompatible with any other lock types
- (U) locks are compatible with (S) but incompatible with (U)



Isolation Level	(S) Locks behavior	Table Hint
Read Uncommitted	(S) locks not acquired	NOLOCK
Read Committed	Acquired and released immediately	READCOMMITTED
Repeatable Read	Held till end of transaction	REPEATABLEREAD
Serializable	Range locks held till end of transaction	HOLDLOCK

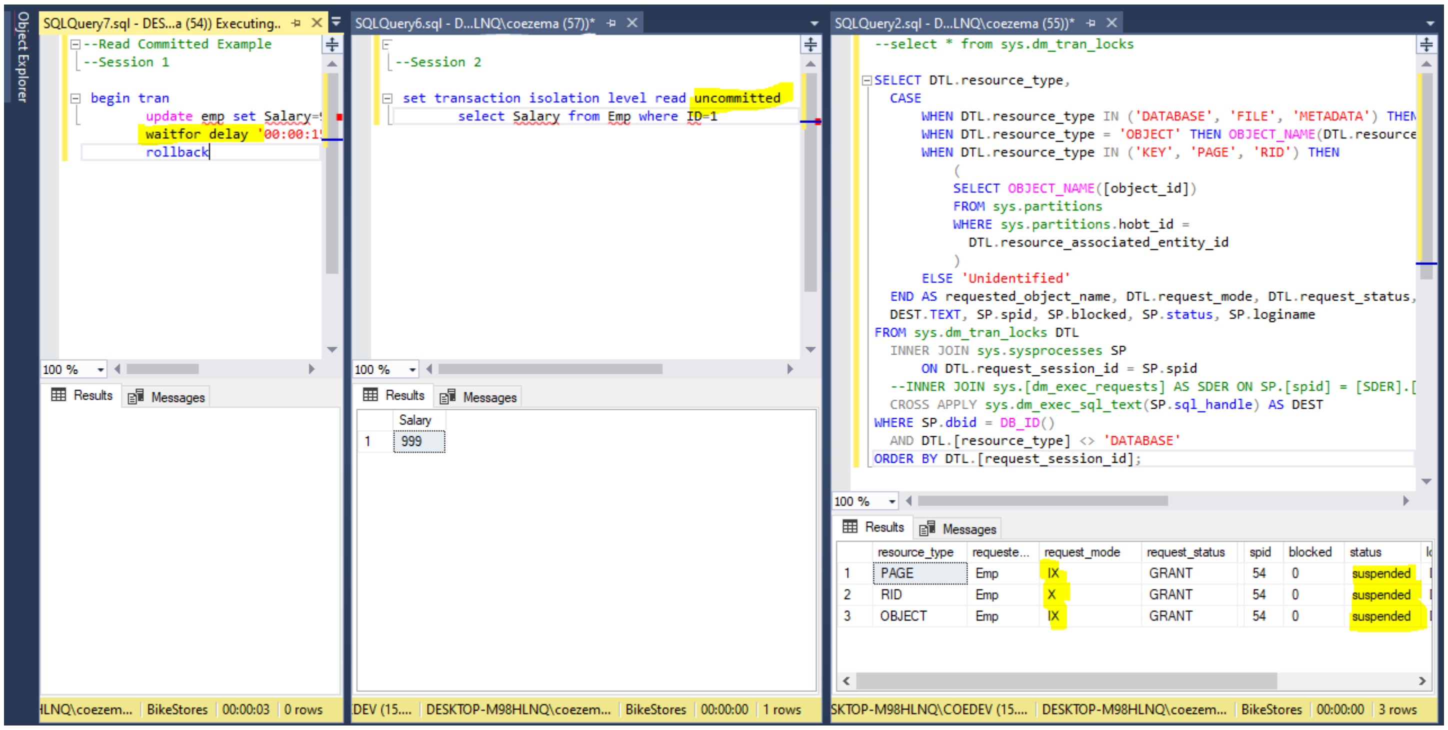
<u>Isolation Levels Examples:</u>

1. **<u>Read Uncommitted</u>** – Read Uncommitted is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transaction, thereby allowing dirty reads. In this level, transactions are not isolated from each other.

   If any table is updated(insert or update or delete) under a transaction and same transaction is not completed that is not committed or roll backed then uncommitted values will display (Dirty Read) in select query of "Read Uncommitted" isolation transaction sessions. There won't be any delay in select query execution because this transaction level does not wait for committed values on table.

   In the below, select query in Session2 executes after update Emp table in transaction and before transaction rolled back. Hence 999 is returned instead of 1000 which is the default value.

   We can note that there is an exclusive lock for the row ID and Intent exclusive lock for the page and object resource types. We can also see that the status has been suspended due to the wait time delay in the session 1 and the session 2 still giving out the updated value for the row.


Object Explorer
SQLQuery7.sql - DES...a (54)) Executing..
SQLQuery6.sql - D...LNQ\coezema (57))*
SQLQuery2.sql - D...LNQ\coezema (55))*
100 %
Results
Messages

```
--Read Committed Example
--Session 1

begin tran
        update emp set Salary=
        waitfor delay '00:00:1
        rollback
```

```
HLNQ\coezem... | BikeStores | 00:00:03 | 0 rows
```

```
--Session 2

set transaction isolation level read uncommitted
        select Salary from Emp where ID=1
```

	Salary
1	999

```
DEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 1 rows
```

```
--select * from sys.dm_tran_locks

SELECT DTL.resource_type,
  CASE
      WHEN DTL.resource_type IN ('DATABASE', 'FILE', 'METADATA') THEN
      WHEN DTL.resource_type = 'OBJECT' THEN OBJECT_NAME(DTL.resource
      WHEN DTL.resource_type IN ('KEY', 'PAGE', 'RID') THEN
         (
         SELECT OBJECT_NAME([object_id])
         FROM sys.partitions
         WHERE sys.partitions.hobt_id =
           DTL.resource_associated_entity_id
         )
      ELSE 'Unidentified'
  END AS requested_object_name, DTL.request_mode, DTL.request_status,
  DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
  INNER JOIN sys.sysprocesses SP
      ON DTL.request_session_id = SP.spid
  --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SDER].[
  CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
WHERE SP.dbid = DB_ID()
  AND DTL.[resource_type] <> 'DATABASE'
ORDER BY DTL.[request_session_id];
```

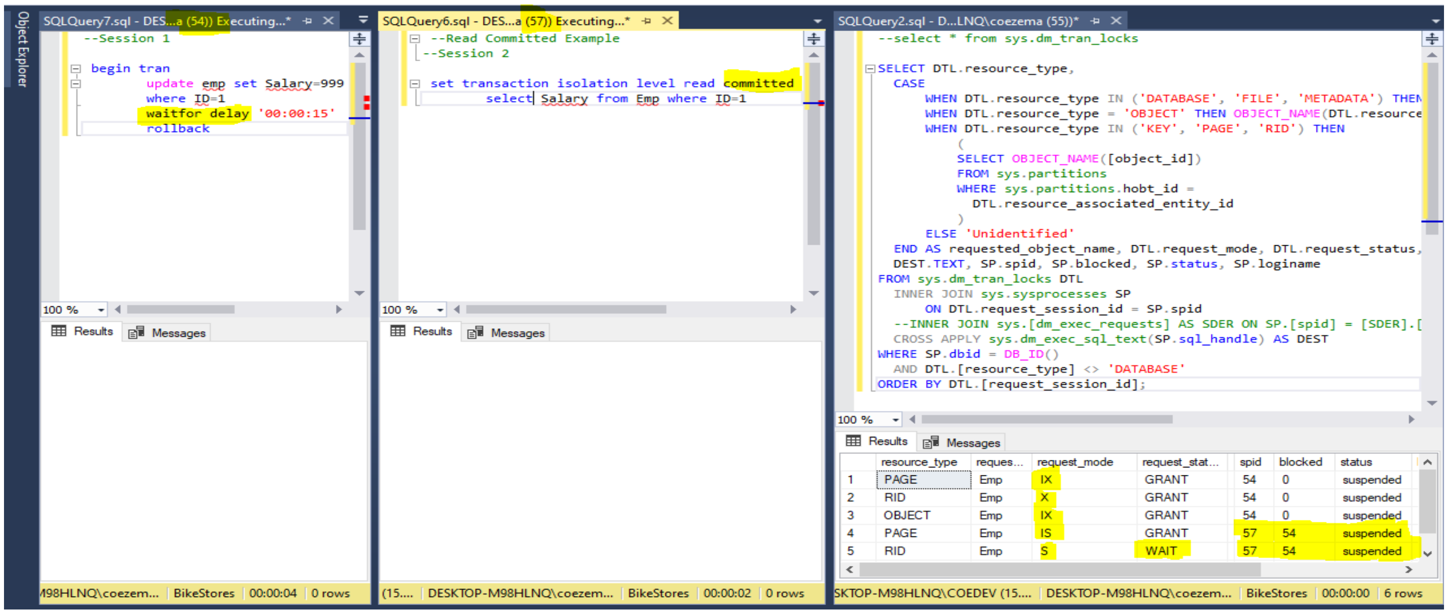
	resource_type	requeste...	request_mode	request_status	spid	blocked	status
1	PAGE	Emp	IX	GRANT	54	0	suspended
2	RID	Emp	X	GRANT	54	0	suspended
3	OBJECT	Emp	IX	GRANT	54	0	suspended

```
SKTOP-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 3 rows
```

2. **<u>Read Committed</u>** – This isolation level guarantees that any data read is committed at that moment it is read. Thus, it does not allow dirty read. The transaction holds a read or write lock on the current row, and thus prevent other transactions from reading, updating or deleting it.

   In select query it will take only committed values of table. If any transaction is opened and incomplete on table in other sessions then select query will wait till no transactions are pending on same table.

As seen below, in second session, it returns the result only after execution of complete transaction in first session because of the lock on Emp table. We have used wait command to delay 15 seconds after updating the Emp table in transaction.



SQLQuery7.sql - DES...a (54)) Executing...*

```
--Session 1

begin tran
        update emp set Salary=999
        where ID=1
        waitfor delay '00:00:15'
        rollback
```

M98HLNQ\coezem... | BikeStores | 00:00:04 | 0 rows

SQLQuery6.sql - DES...a (57)) Executing...*

```
--Read Committed Example
--Session 2

set transaction isolation level read committed
        select Salary from Emp where ID=1
```

(15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:02 | 0 rows

SQLQuery2.sql - D...LNQ\coezema (55))*

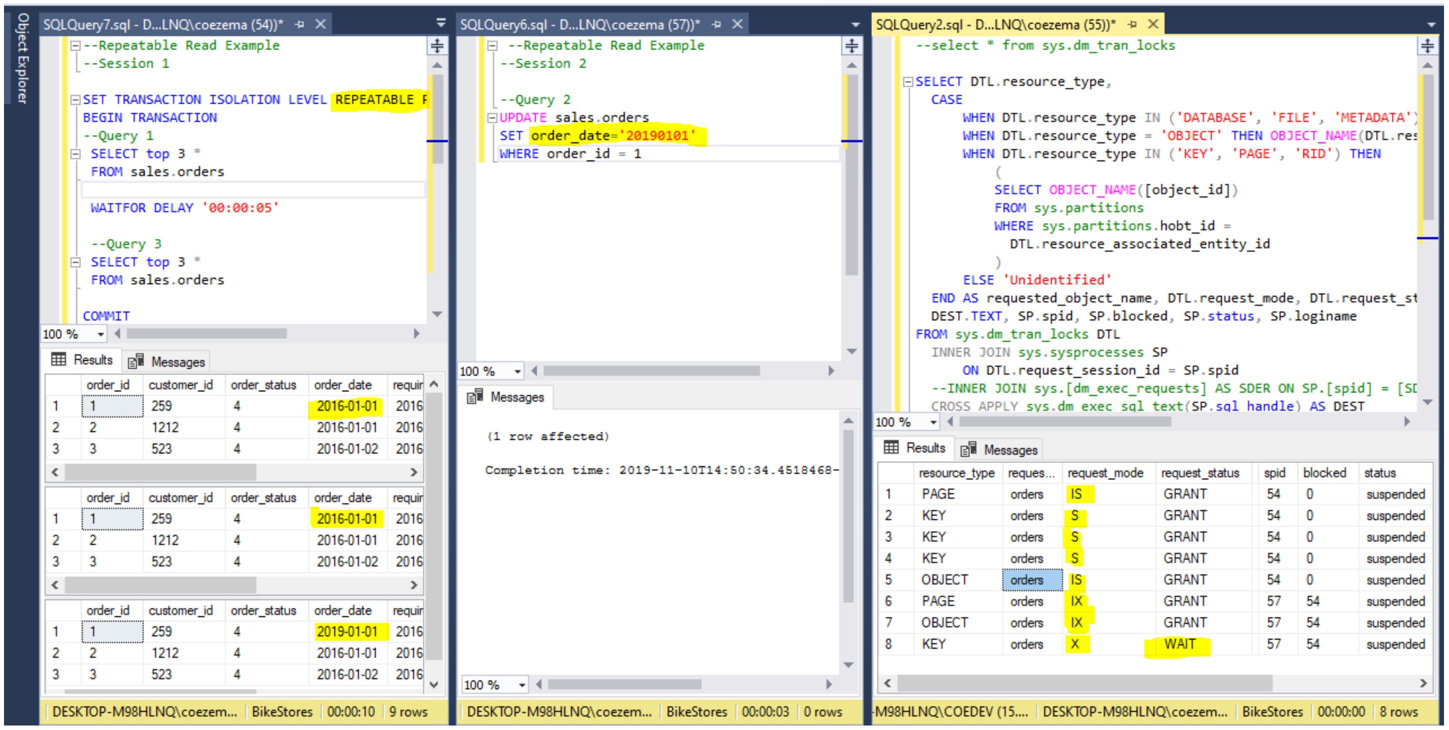
```
--select * from sys.dm_tran_locks

SELECT DTL.resource_type,
  CASE
      WHEN DTL.resource_type IN ('DATABASE', 'FILE', 'METADATA') THEN
      WHEN DTL.resource_type = 'OBJECT' THEN OBJECT_NAME(DTL.resource
      WHEN DTL.resource_type IN ('KEY', 'PAGE', 'RID') THEN
          (
          SELECT OBJECT_NAME([object_id])
          FROM sys.partitions
          WHERE sys.partitions.hobt_id =
            DTL.resource_associated_entity_id
          )
      ELSE 'Unidentified'
  END AS requested_object_name, DTL.request_mode, DTL.request_status,
  DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
  INNER JOIN sys.sysprocesses SP
      ON DTL.request_session_id = SP.spid
  --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SDER].[
  CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
WHERE SP.dbid = DB_ID()
  AND DTL.[resource_type] <> 'DATABASE'
ORDER BY DTL.[request_session_id];
```

	resource_type	reques...	request_mode	request_stat...	spid	blocked	status
1	PAGE	Emp	IX	GRANT	54	0	suspended
2	RID	Emp	X	GRANT	54	0	suspended
3	OBJECT	Emp	IX	GRANT	54	0	suspended
4	PAGE	Emp	IS	GRANT	57	54	suspended
5	RID	Emp	S	WAIT	57	54	suspended

SKTOP-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 6 rows

3. **Repeatable Read** – This is the most restrictive isolation level. The transaction holds read locks on all rows it references and writes locks on all rows it inserts, updates, or deletes. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.

   Update command in session 2 will wait till session 1 transaction is completed because sales order table row with order date=20190101 has locked in session1 transaction. In session 2 will execute without any delay because it has insert query for new entry. This isolation level allows to insert new data but does not allow to modify data that is used in select query executed in transaction. We can notice two results displayed in Session 1 have different number of row count(1 row extra in second result set).


SQLQuery7.sql - D...LNQ\coezema (54))*

```sql
--Repeatable Read Example
--Session 1

SET TRANSACTION ISOLATION LEVEL REPEATABLE R
BEGIN TRANSACTION
--Query 1
 SELECT top 3 *
 FROM sales.orders

 WAITFOR DELAY '00:00:05'

 --Query 3
 SELECT top 3 *
 FROM sales.orders

COMMIT
```

	order_id	customer_id	order_status	order_date	requir
1	1	259	4	2016-01-01	2016
2	2	1212	4	2016-01-01	2016
3	3	523	4	2016-01-02	2016

	order_id	customer_id	order_status	order_date	requir
1	1	259	4	2016-01-01	2016
2	2	1212	4	2016-01-01	2016
3	3	523	4	2016-01-02	2016

	order_id	customer_id	order_status	order_date	requir
1	1	259	4	2019-01-01	2016
2	2	1212	4	2016-01-01	2016
3	3	523	4	2016-01-02	2016

DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:10 | 9 rows

SQLQuery6.sql - D...LNQ\coezema (57))*

```sql
--Repeatable Read Example
--Session 2

--Query 2
UPDATE sales.orders
SET order_date='20190101'
WHERE order_id = 1
```

```
(1 row affected)

Completion time: 2019-11-10T14:50:34.4518468-
```

DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:03 | 0 rows

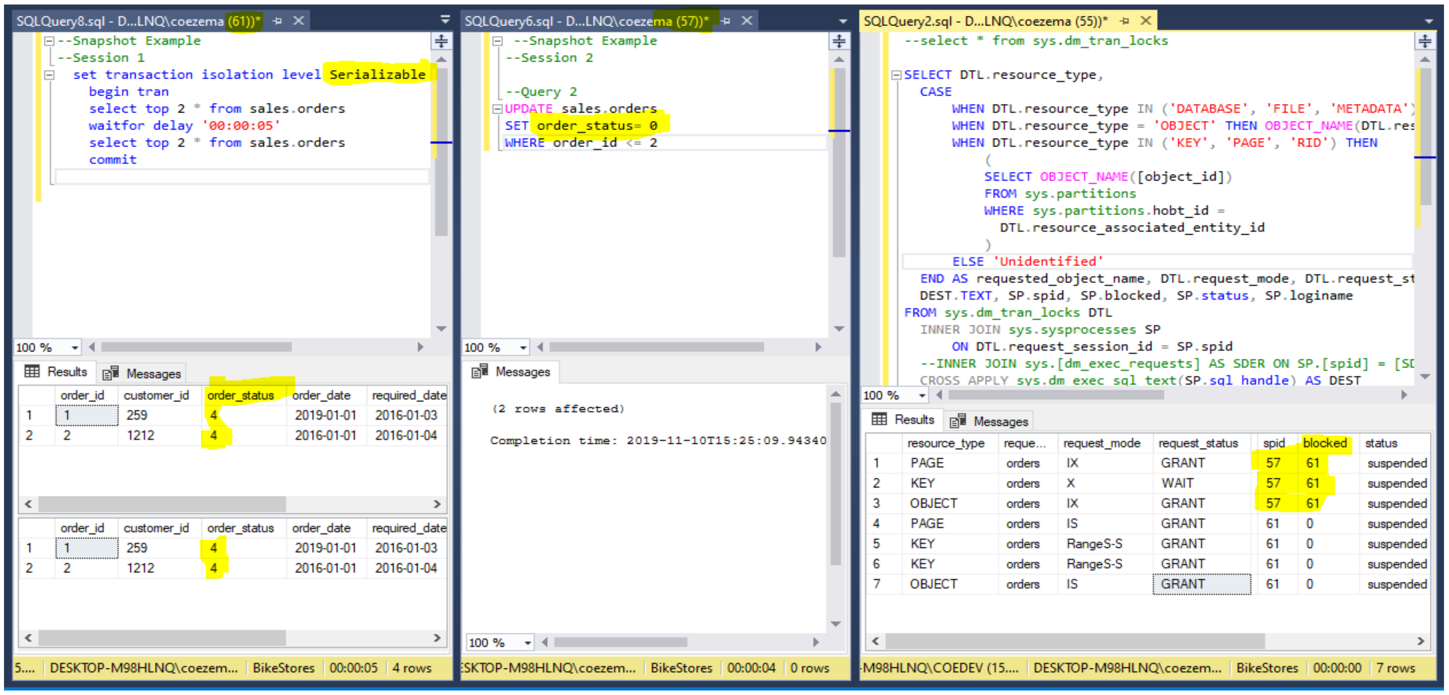
SQLQuery2.sql - D...LNQ\coezema (55))*

```sql
--select * from sys.dm_tran_locks

SELECT DTL.resource_type,
  CASE
      WHEN DTL.resource_type IN ('DATABASE', 'FILE', 'METADATA')
      WHEN DTL.resource_type = 'OBJECT' THEN OBJECT_NAME(DTL.res
      WHEN DTL.resource_type IN ('KEY', 'PAGE', 'RID') THEN
          (
          SELECT OBJECT_NAME([object_id])
          FROM sys.partitions
          WHERE sys.partitions.hobt_id =
            DTL.resource_associated_entity_id
          )
      ELSE 'Unidentified'
  END AS requested_object_name, DTL.request_mode, DTL.request_st
  DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
  INNER JOIN sys.sysprocesses SP
      ON DTL.request_session_id = SP.spid
  --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SD
  CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
```

	resource_type	reques...	request_mode	request_status	spid	blocked	status
1	PAGE	orders	IS	GRANT	54	0	suspended
2	KEY	orders	S	GRANT	54	0	suspended
3	KEY	orders	S	GRANT	54	0	suspended
4	KEY	orders	S	GRANT	54	0	suspended
5	OBJECT	orders	IS	GRANT	54	0	suspended
6	PAGE	orders	IX	GRANT	57	54	suspended
7	OBJECT	orders	IX	GRANT	57	54	suspended
8	KEY	orders	X	WAIT	57	54	suspended

-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 8 rows

4. **Serializable** – This is the Highest isolation level. Serializable Isolation is similar to Repeatable Read Isolation but the difference is it prevents Phantom Read. This works based on range lock. If table has index then it locks records based on index range used in WHERE clause (like where ID between 1 and 3). If table doesn't have index then it locks complete table.
In the below example, Complete sales order table will be locked during the transaction in Session 1. Unlike "Repeatable Read", insert query in Session 2 will wait till session 1 execution is completed. Hence Phantom read is prevented and both queries in session 1 will display same number of rows.


SQLQuery8.sql - D...LNQ\coezema (61))*

```sql
--Snapshot Example
--Session 1
  set transaction isolation level Serializable
    begin tran
    select top 2 * from sales.orders
    waitfor delay '00:00:05'
    select top 2 * from sales.orders
    commit
```

	order_id	customer_id	order_status	order_date	required_date
1	1	259	4	2019-01-01	2016-01-03
2	2	1212	4	2016-01-01	2016-01-04

	order_id	customer_id	order_status	order_date	required_date
1	1	259	4	2019-01-01	2016-01-03
2	2	1212	4	2016-01-01	2016-01-04

5.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:05 | 4 rows

SQLQuery6.sql - D...LNQ\coezema (57))*

```sql
--Snapshot Example
--Session 2

--Query 2
UPDATE sales.orders
SET order_status= 0
WHERE order_id <= 2
```

```
(2 rows affected)

Completion time: 2019-11-10T15:25:09.94340
```

SKTOP-M98HLNQ\coezem... | BikeStores | 00:00:04 | 0 rows

SQLQuery2.sql - D...LNQ\coezema (55))*

```sql
--select * from sys.dm_tran_locks

SELECT DTL.resource_type,
  CASE
      WHEN DTL.resource_type IN ('DATABASE', 'FILE', 'METADATA')
      WHEN DTL.resource_type = 'OBJECT' THEN OBJECT_NAME(DTL.res
      WHEN DTL.resource_type IN ('KEY', 'PAGE', 'RID') THEN
          (
          SELECT OBJECT_NAME([object_id])
          FROM sys.partitions
          WHERE sys.partitions.hobt_id =
            DTL.resource_associated_entity_id
          )
      ELSE 'Unidentified'
  END AS requested_object_name, DTL.request_mode, DTL.request_st
  DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
  INNER JOIN sys.sysprocesses SP
      ON DTL.request_session_id = SP.spid
  --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SD
  CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
```

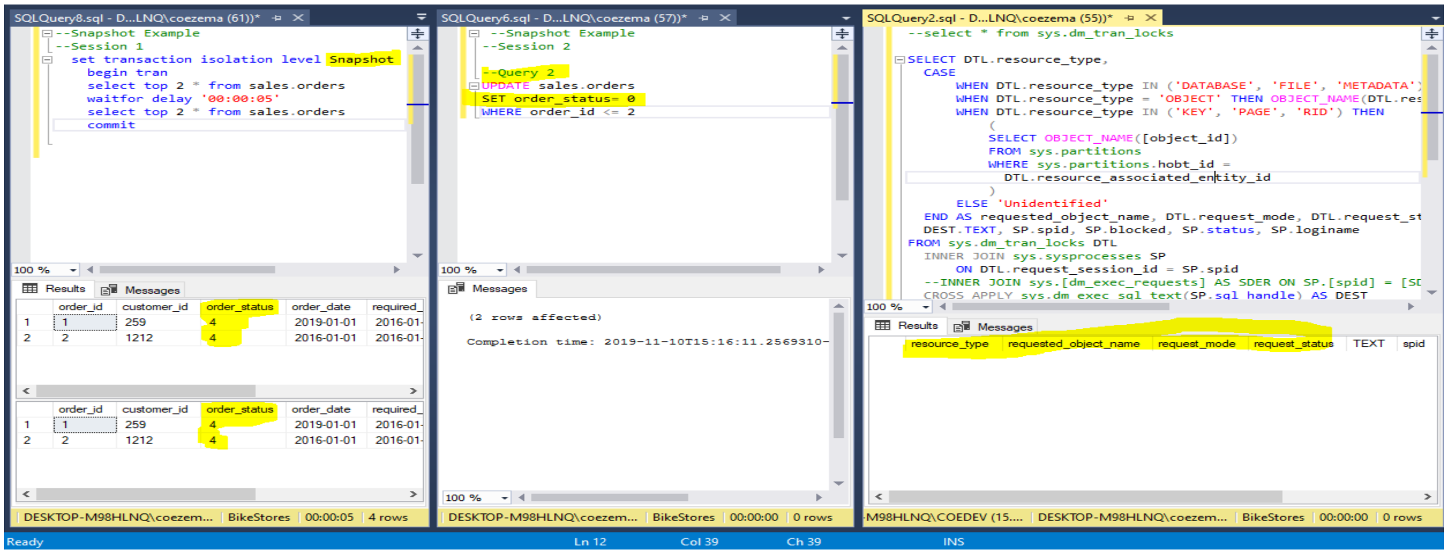
	resource_type	reque...	request_mode	request_status	spid	blocked	status
1	PAGE	orders	IX	GRANT	57	61	suspended
2	KEY	orders	X	WAIT	57	61	suspended
3	OBJECT	orders	IX	GRANT	57	61	suspended
4	PAGE	orders	IS	GRANT	61	0	suspended
5	KEY	orders	RangeS-S	GRANT	61	0	suspended
6	KEY	orders	RangeS-S	GRANT	61	0	suspended
7	OBJECT	orders	IS	GRANT	61	0	suspended

-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 7 rows

Compare with row versioning (snapshot):

Snapshot isolation is similar to Serializable isolation. The difference is Snapshot does not hold lock on table during the transaction so table can be modified in other sessions. Snapshot isolation maintains versioning in Tempdb for old data in case of any data modification occurs in other sessions then existing transaction displays the old data from Tempdb.

Running both queries simultaneously, Session 2 queries will be executed in parallel as transaction in session 1 won't lock the table Emp. Hence we don’t any block, lock options in the dynamic management view table when queried during the execution of these queries.


SQLQuery8.sql - D...LNQ\coezema (61))*

```
--Snapshot Example
--Session 1
  set transaction isolation level Snapshot
    begin tran
    select top 2 * from sales.orders
    waitfor delay '00:00:05'
    select top 2 * from sales.orders
    commit
```

	order_id	customer_id	order_status	order_date	required_
1	1	259	4	2019-01-01	2016-01-
2	2	1212	4	2016-01-01	2016-01-

	order_id	customer_id	order_status	order_date	required_
1	1	259	4	2019-01-01	2016-01-
2	2	1212	4	2016-01-01	2016-01-

DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:05 | 4 rows

SQLQuery6.sql - D...LNQ\coezema (57))*

```
 --Snapshot Example
--Session 2

--Query 2
UPDATE sales.orders
SET order_status= 0
WHERE order_id <= 2
```

```
(2 rows affected)

Completion time: 2019-11-10T15:16:11.2569310-
```

DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 0 rows

SQLQuery2.sql - D...LNQ\coezema (55))*

```
--select * from sys.dm_tran_locks

SELECT DTL.resource_type,
  CASE
      WHEN DTL.resource_type IN ('DATABASE', 'FILE', 'METADATA')
      WHEN DTL.resource_type = 'OBJECT' THEN OBJECT_NAME(DTL.res
      WHEN DTL.resource_type IN ('KEY', 'PAGE', 'RID') THEN
        (
        SELECT OBJECT_NAME([object_id])
        FROM sys.partitions
        WHERE sys.partitions.hobt_id =
          DTL.resource_associated_entity_id
        )
      ELSE 'Unidentified'
  END AS requested_object_name, DTL.request_mode, DTL.request_st
  DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
  INNER JOIN sys.sysprocesses SP
      ON DTL.request_session_id = SP.spid
  --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SD
  CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
```

resource_type	requested_object_name	request_mode	request_status	TEXT	spid

M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 0 rows

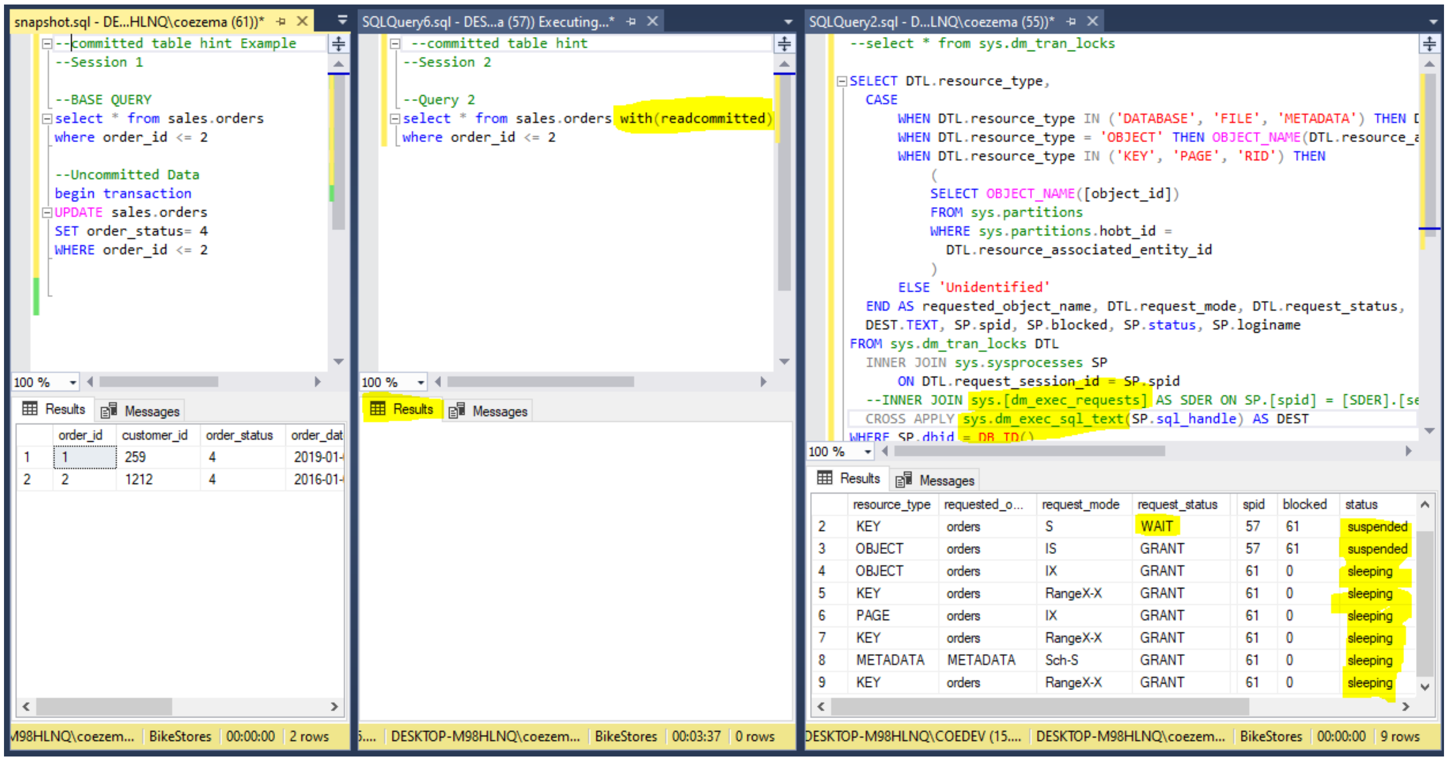
Ready Ln 12 Col 39 Ch 39 INS

Table Hints:

1. Using ReadCommitted hint:

This is the default table locking behavior used by SQL Server. This type of isolation uses shared locks when reading data to ensure that only data that is committed to the database have been read. These locks are used because updates required exclusive locks on the data that they are modifying, which block any readers that try to use shared locks. For most general querying purposes, this isolation level does a good job of balancing database consistency and concurrency.

In the below example, we are trying to read the committed data from the uncommitted query in session 1. We can see that there is no result for the query using the read committed table hint. In the DMV window we can see how the query has been suspended by the session 1 as there is no COMMIT OR ROLLBACK command to the query.



snapshot.sql - DE...HLNQ\coezema (61))*

```sql
--committed table hint Example
--Session 1

--BASE QUERY
select * from sales.orders
where order_id <= 2

--Uncommitted Data
begin transaction
UPDATE sales.orders
SET order_status= 4
WHERE order_id <= 2
```

	order_id	customer_id	order_status	order_dat
1	1	259	4	2019-01-
2	2	1212	4	2016-01-

```
M98HLNQ\coezem... | BikeStores | 00:00:00 | 2 rows
```

SQLQuery6.sql - DES...a (57)) Executing...*

```sql
--committed table hint
--Session 2

--Query 2
select * from sales.orders with(readcommitted)
where order_id <= 2
```

```
5.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:03:37 | 0 rows
```

SQLQuery2.sql - D...LNQ\coezema (55))*

```sql
--select * from sys.dm_tran_locks

SELECT DTL.resource_type,
  CASE
      WHEN DTL.resource_type IN ('DATABASE', 'FILE', 'METADATA') THEN D
      WHEN DTL.resource_type = 'OBJECT' THEN OBJECT_NAME(DTL.resource_a
      WHEN DTL.resource_type IN ('KEY', 'PAGE', 'RID') THEN
          (
          SELECT OBJECT_NAME([object_id])
          FROM sys.partitions
          WHERE sys.partitions.hobt_id =
            DTL.resource_associated_entity_id
          )
      ELSE 'Unidentified'
  END AS requested_object_name, DTL.request_mode, DTL.request_status,
  DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
  INNER JOIN sys.sysprocesses SP
      ON DTL.request_session_id = SP.spid
  --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SDER].[se
  CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
WHERE SP.dbid = DB_ID()
```

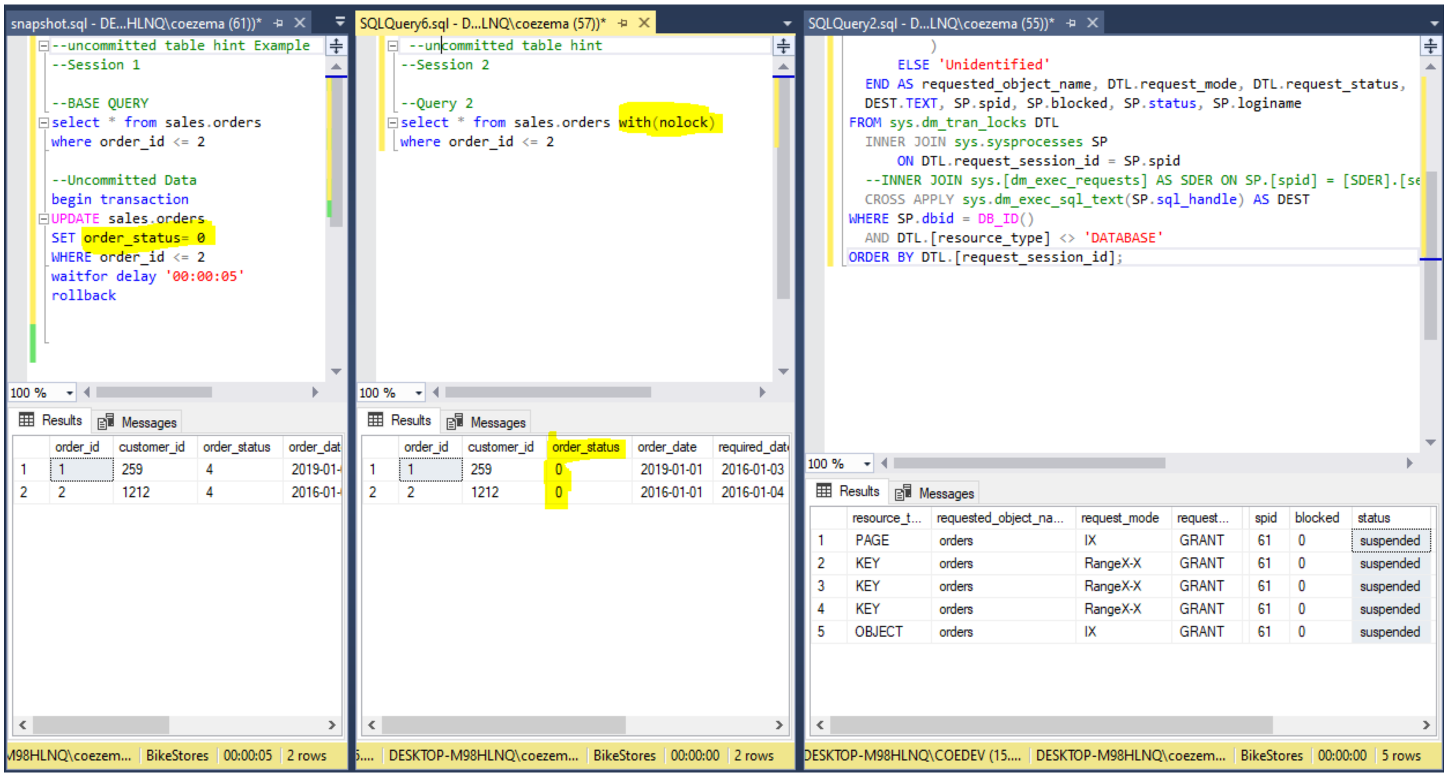
	resource_type	requested_o...	request_mode	request_status	spid	blocked	status
2	KEY	orders	S	WAIT	57	61	suspended
3	OBJECT	orders	IS	GRANT	57	61	suspended
4	OBJECT	orders	IX	GRANT	61	0	sleeping
5	KEY	orders	RangeX-X	GRANT	61	0	sleeping
6	PAGE	orders	IX	GRANT	61	0	sleeping
7	KEY	orders	RangeX-X	GRANT	61	0	sleeping
8	METADATA	METADATA	Sch-S	GRANT	61	0	sleeping
9	KEY	orders	RangeX-X	GRANT	61	0	sleeping

```
DESKTOP-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 9 rows
```

2. <u>Using NOLOCK hint:</u>

The WITH (NOLOCK) table hint is used to override the default transaction isolation level of the table or the tables within the view in a specific query, by allowing the user to retrieve the data without being affected by the locks, on the requested data, due to another process that is changing it. In this way, the query will consume less memory in holding locks against that data. In addition to that, no deadlock will occur against the queries, that are requesting the same data from that table, allowing a higher level of concurrency due to a lower footprint. In other words, the WITH (NOLOCK) table hint retrieves the rows without waiting for the other queries, that are reading or modifying the same data, to finish its processing. This is similar to the READ UNCOMMITTED transaction isolation level, that allows the query to see the data changes before committing the transaction that is changing it.

In the below example, we are trying to read the uncommitted data from the query in session 1. We can see that there is the result for the query using the read uncommitted table hint. In the DMV window we can see how the query has been suspended by the session 1 as there is no COMMIT OR ROLLBACK command to the query.


snapshot.sql - DE...HLNQ\coezema (61))*

```sql
--uncommitted table hint Example
--Session 1

--BASE QUERY
select * from sales.orders
where order_id <= 2

--Uncommitted Data
begin transaction
UPDATE sales.orders
SET order_status= 0
WHERE order_id <= 2
waitfor delay '00:00:05'
rollback
```

	order_id	customer_id	order_status	order_dat
1	1	259	4	2019-01-
2	2	1212	4	2016-01-

```
M98HLNQ\coezem... | BikeStores | 00:00:05 | 2 rows
```

SQLQuery6.sql - D...LNQ\coezema (57))*

```sql
--uncommitted table hint
--Session 2

--Query 2
select * from sales.orders with(nolock)
where order_id <= 2
```

	order_id	customer_id	order_status	order_date	required_dat
1	1	259	0	2019-01-01	2016-01-03
2	2	1212	0	2016-01-01	2016-01-04

```
5.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 2 rows
```

SQLQuery2.sql - D...LNQ\coezema (55))*

```sql
            )
        ELSE 'Unidentified'
    END AS requested_object_name, DTL.request_mode, DTL.request_status,
    DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
    INNER JOIN sys.sysprocesses SP
        ON DTL.request_session_id = SP.spid
    --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SDER].[se
    CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
WHERE SP.dbid = DB_ID()
    AND DTL.[resource_type] <> 'DATABASE'
ORDER BY DTL.[request_session_id];
```

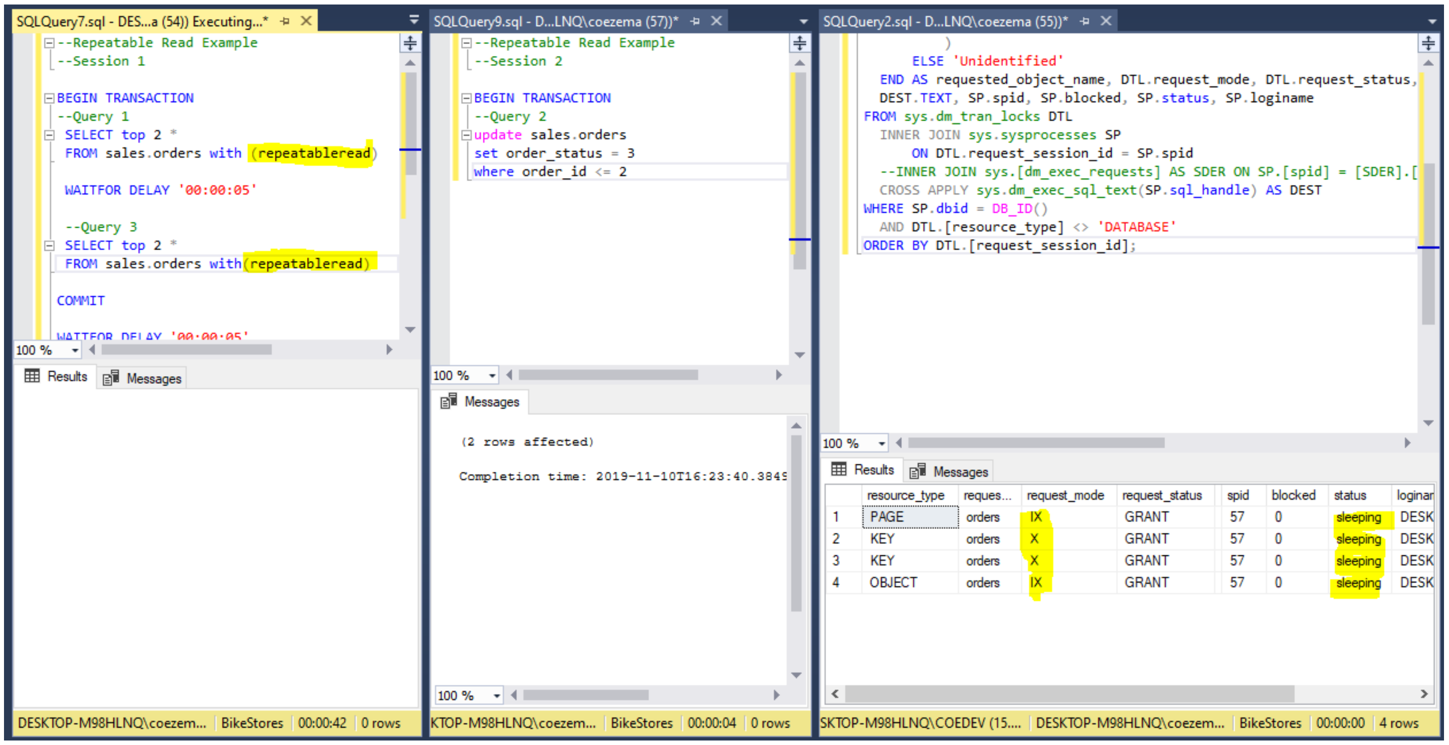
	resource_t...	requested_object_na...	request_mode	request...	spid	blocked	status
1	PAGE	orders	IX	GRANT	61	0	suspended
2	KEY	orders	RangeX-X	GRANT	61	0	suspended
3	KEY	orders	RangeX-X	GRANT	61	0	suspended
4	KEY	orders	RangeX-X	GRANT	61	0	suspended
5	OBJECT	orders	IX	GRANT	61	0	suspended

```
DESKTOP-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 5 rows
```

3. Using RepeatableRead hint:

The REPEATABLEREAD table hint can deter Non-Repeatable reads. The following example shows this table hint in action. The query from Session 2, which uses the REPEATABLEREAD table hint, causes problems. When this table hint is issued, it guarantees that, for the records that were already returned by the transaction, these records cannot be changed by outside transactions. This ensures that these records can be read repeatedly, and the data will not have changed. The statement above **causes a deadlock** scenario, and the SQL Server database engine has picked one of the sessions as the **deadlock victim**.

While the REPEATABLEREAD isolation level prevents a Non-Repeatable read, it does not prevent phantom reads. It guards the data that has already been returned by the transaction, but it does not guard against new records in the previously returned result set. This phantom read scenario occurs when one transaction has a range or rows of an entire table locked in a transaction, and a separate transaction inserts a row into that locked range or rows in the table. Any subsequent reads by the first transaction result in the ability to see the new row, even though it was not present in the original read.


SQLQuery7.sql - DES...a (54)) Executing...*

```sql
--Repeatable Read Example
--Session 1

BEGIN TRANSACTION
--Query 1
 SELECT top 2 *
 FROM sales.orders with (repeatableread)

 WAITFOR DELAY '00:00:05'

 --Query 3
 SELECT top 2 *
 FROM sales.orders with(repeatableread)

COMMIT

WAITFOR DELAY '00:00:05'
```

DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:42 | 0 rows

SQLQuery9.sql - D...LNQ\coezema (57))*

```sql
--Repeatable Read Example
--Session 2

BEGIN TRANSACTION
--Query 2
update sales.orders
set order_status = 3
where order_id <= 2
```

```
(2 rows affected)

Completion time: 2019-11-10T16:23:40.3845
```

KTOP-M98HLNQ\coezem... | BikeStores | 00:00:04 | 0 rows

SQLQuery2.sql - D...LNQ\coezema (55))*

```sql
            )
        ELSE 'Unidentified'
    END AS requested_object_name, DTL.request_mode, DTL.request_status,
    DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
    INNER JOIN sys.sysprocesses SP
        ON DTL.request_session_id = SP.spid
    --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SDER].[
    CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
WHERE SP.dbid = DB_ID()
    AND DTL.[resource_type] <> 'DATABASE'
ORDER BY DTL.[request_session_id];
```

	resource_type	reques...	request_mode	request_status	spid	blocked	status	loginar
1	PAGE	orders	IX	GRANT	57	0	sleeping	DESK
2	KEY	orders	X	GRANT	57	0	sleeping	DESK
3	KEY	orders	X	GRANT	57	0	sleeping	DESK
4	OBJECT	orders	IX	GRANT	57	0	sleeping	DESK

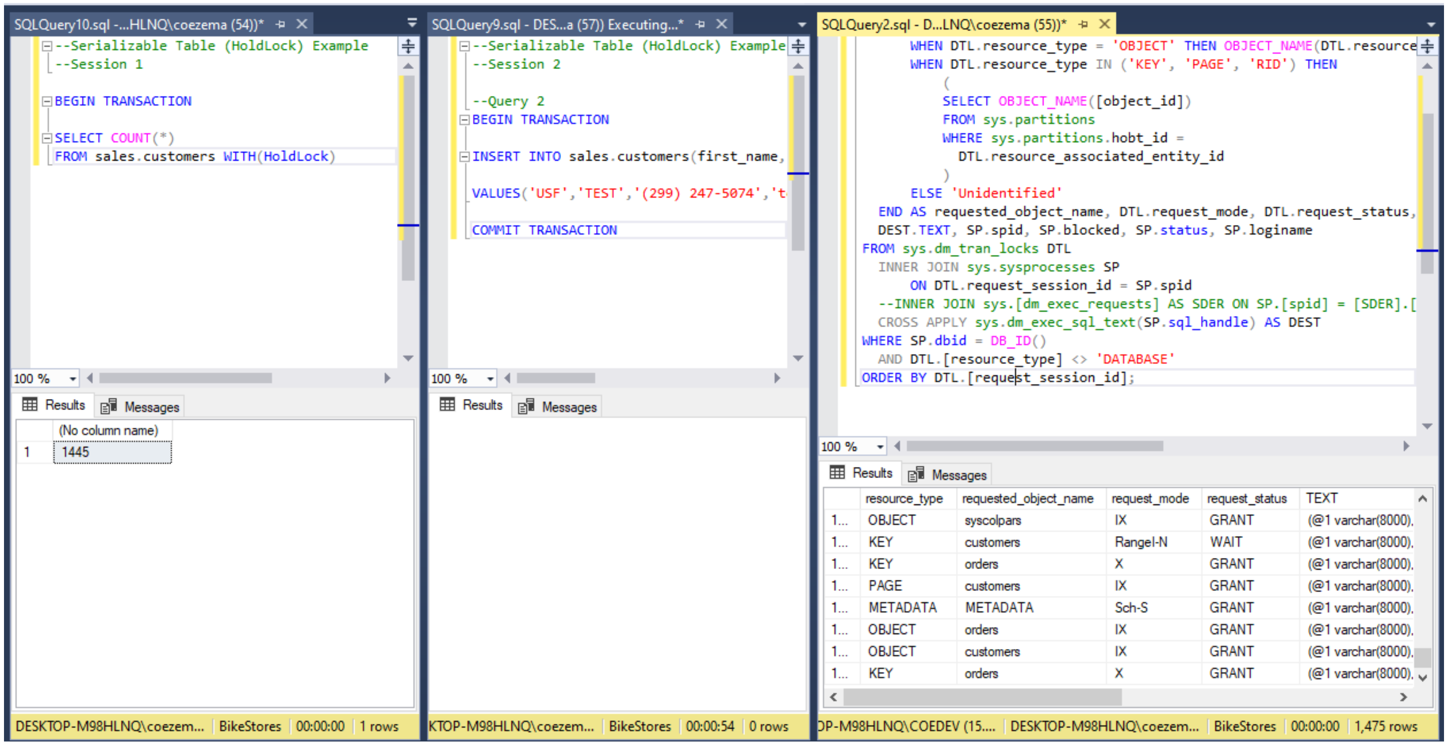
SKTOP-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 4 rows

4. Using HOLDLOCK hint:

The SERIALIZABLE table hint is at the other end of the spectrum from the NOLOCK table hint. When we use the SERIALIZABLE table hint (also known as HOLDLOCK), it guarantees that no other transaction can modify or read uncommitted data in the current transaction. In other words, transactions must wait for other transactions to complete before completing their work. This drastically limits database concurrency and puts a premium on database consistency.

This session tries to insert a record into the SalesOrder table, which is currently locked from Session1. Because of the blocking, this session will wait continuously until the transaction from Session1 has completed. This table hint prevents all other concurrency side effects that I previously discussed -- dirty reads, Non-Repeatable reads, and phantom reads. The price you pay for this high data consistency is that other transactions must wait for the locking transaction to complete. This can drastically affect performance due to the reduced concurrency.

In the below query, Session 2 tries to insert a new customer record into the sales.customer table, which is currently locked from Session 1. Because of the blocking, session 2 will wait continuously, hence the blank space in result until the transaction from Session 1 has completed. HoldLock table hint prevents all other concurrency side effects like dirty reads, Non-Repeatable reads, and phantom reads. Although, this allows for high data consistency, the drawback is that other transactions must wait for the locking transaction to complete.



SQLQuery10.sql -...HLNQ\coezema (54))*

```
--Serializable Table (HoldLock) Example
--Session 1

BEGIN TRANSACTION

SELECT COUNT(*)
FROM sales.customers WITH(HoldLock)
```

	(No column name)
1	1445

DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 1 rows

SQLQuery9.sql - DES...a (57)) Executing...*

```
--Serializable Table (HoldLock) Example
--Session 2

--Query 2
BEGIN TRANSACTION

INSERT INTO sales.customers(first_name,

VALUES('USF','TEST','(299) 247-5074','t

COMMIT TRANSACTION
```

KTOP-M98HLNQ\coezem... | BikeStores | 00:00:54 | 0 rows

SQLQuery2.sql - D...LNQ\coezema (55))*

```
        WHEN DTL.resource_type = 'OBJECT' THEN OBJECT_NAME(DTL.resource
        WHEN DTL.resource_type IN ('KEY', 'PAGE', 'RID') THEN
            (
            SELECT OBJECT_NAME([object_id])
            FROM sys.partitions
            WHERE sys.partitions.hobt_id =
              DTL.resource_associated_entity_id
            )
        ELSE 'Unidentified'
    END AS requested_object_name, DTL.request_mode, DTL.request_status,
    DEST.TEXT, SP.spid, SP.blocked, SP.status, SP.loginame
FROM sys.dm_tran_locks DTL
    INNER JOIN sys.sysprocesses SP
        ON DTL.request_session_id = SP.spid
    --INNER JOIN sys.[dm_exec_requests] AS SDER ON SP.[spid] = [SDER].[
    CROSS APPLY sys.dm_exec_sql_text(SP.sql_handle) AS DEST
WHERE SP.dbid = DB_ID()
    AND DTL.[resource_type] <> 'DATABASE'
ORDER BY DTL.[request_session_id];
```

	resource_type	requested_object_name	request_mode	request_status	TEXT
1...	OBJECT	syscolpars	IX	GRANT	(@1 varchar(8000),
1...	KEY	customers	RangeI-N	WAIT	(@1 varchar(8000),
1...	KEY	orders	X	GRANT	(@1 varchar(8000),
1...	PAGE	customers	IX	GRANT	(@1 varchar(8000),
1...	METADATA	METADATA	Sch-S	GRANT	(@1 varchar(8000),
1...	OBJECT	orders	IX	GRANT	(@1 varchar(8000),
1...	OBJECT	customers	IX	GRANT	(@1 varchar(8000),
1...	KEY	orders	X	GRANT	(@1 varchar(8000),

DP-M98HLNQ\COEDEV (15.... | DESKTOP-M98HLNQ\coezem... | BikeStores | 00:00:00 | 1,475 rows

<u>**Dead Lock Resolution:**</u>

The nice thing about deadlocks is that SQL Server automatically detects and resolves them. To resolve a deadlock, SQL Server has to rollback the cheapest of the 2 transactions. In the context of SQL Server, the cheapest transaction is the transaction that has written the fewer bytes to the transaction log.

SQL Server implements the deadlock detection in a background process called the Deadlock Monitor. This background process runs every 5 seconds and checks the current locking situation for deadlocks. In the worst case, a deadlock should therefore not last longer than 5 seconds. The query which gets rolled back receives the error number 1205. The good thing about deadlocks is that we can fully recover from that error situation without any user interaction. The following steps can help us to recover from a deadlock:

- Check for error number 1205, when an exception is thrown
- Pause the application briefly to give the other query time to complete its transaction and release its acquired locks
- Resubmit the query, which was rolled back by SQL Server

After the resubmission of the query, the query should continue without any problems, because the other blocking query will have already finished its transaction. Of course we should keep track of re-occurrence of deadlocks, so that we do not retry the transaction over and over again.